Widening with Thresholds for Programs with Complex Control Graphs

Lies Lakhdar-Chaouch, Bertrand Jeannet, Alain Girault

Widening with Thresholds for Programs with Complex Control Graphs*

Lies Lakhdar-Chaouch, Bertrand Jeannet, and Alain Girault

INRIA

Abstract. The precision of an analysis based on abstract interpretation does not only depend on the abstract domain, but also on the solving method. The traditional solution is to solve iteratively abstract fixpoint equations, using extrapolation with a widening operator to make the iterations converge. Unfortunately, this extrapolation often loses crucial information for the analysis goal. A classical technique for improving the precision is "widening with thresholds", which bounds the extrapolation. Its benefit strongly depends on the choice of relevant thresholds. In this paper we propose a semantic-based technique for automatically inferring such thresholds, which applies to any control graph, be it intraprocedural, interprocedural or concurrent, without specific assumptions on the abstract domain. Despite its technical simplicity, our technique is able to infer the relevant thresholds in many practical cases.

1 Introduction and Related Work

Many static analysis problems boil down to the computation of the least solution of a fixpoint equation $X = F(X), X \in C$ where C is a domain of concrete properties, and F a function derived from the semantics of the analyzed program. Abstract Interpretation provides a framework for reducing this problem to the solving of a simpler equation in a domain A of *abstract properties*:

$$Y = G(Y), Y \in A \tag{1}$$

Having performed this *static approximation*, one is left with the problem of solving (1). The paper focuses on this problem. It considers the traditional iterative solving technique with widening and narrowing, and focuses more specifically on the widening with thresholds technique. We first review existing techniques before presenting our approach.

Exact equation solving. Some techniques solves directly (1) in the case where concrete properties are invariants on numerical variables. In [1,2] classes of equations on intervals are identified, for which the least solution can be computed exactly. *Policy iteration* methods solve (1) by solving a succession of simpler equations $Y = G_\pi(Y)$ indexed by a *policy* π [3,4]. However, such approaches are currently restricted to domains that infer bounds on a fixed set of numerical expressions, which excludes for instance the convex polyhedra abstract domain [5] and they do not make obsolete the classical iterative method described next.

* This work was supported by the OpenTLM project (pôle de compétitivité Minalogic).

T. Bultan and P.-A. Hsiung (Eds.): ATVA 2011, LNCS 6996, pp. 492–502, 2011.

Approximate equation solving by widening/narrowing. Under the classical hypothesis the sequence $Y_0 = \bot, Y_{n+1} = G(Y_n)$ converges to $lfp(G)$. However, if A contains infinite ascending sequences, which is the case of the abstract lattices mentioned above, the limit is extrapolated by using a *widening* operator $\nabla : A \times A \to A$. One computes the ascending sequence

$$Y_0 = \bot,\ Y_{n+1} = Y_n \nabla G(Y_n) \tag{2}$$

which converges after a bounded number of iterations to a post-fixpoint $Y_\infty \sqsupseteq lfp(G)$, see Fig. 1. The approximations induced by widening can be partially recovered by performing a few descending iterations defined by the sequence

$$Z_0 = Y_\infty,\ Z_{n+1} = G(Z_n) \tag{3}$$

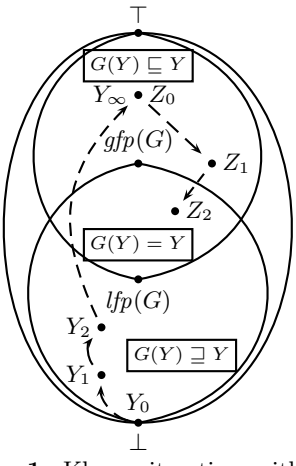


Fig. 1. Kleene iteration with widening and narrowing

This is the most common instance of the concept of *narrowing* (see [6]). For many numerical abstract domains (like octagons [7] or convex polyhedra [5]) the "standard" widening consists in keeping in the result $R = P \nabla Q$ the numerical constraints of P that are still satisfied by Q.

The use of widening adds *dynamic approximations* to the *static approximations* induced by the choice of the abstract domain. Although it is shown in [6] that abstract domains with infinitely ascending sequences can discover properties that simpler abstract domains cannot infer, these dynamic approximations often raise accuracy issues. In particular no widening operator is monotonic. Moreover, as we show in §2, narrowing often fails to recover important information lost by widening, even on simple examples. In particular, if the function G is *extensive* (*i.e.,* $\forall Y \in A, Y \sqsubseteq G(Y)$), narrowing has no effect at all.

Techniques for controlling dynamic approximations. One approach is to improve the standard widening operators [8,9]. Other approaches are more global. For instance, *abstract acceleration* computes precisely with a single formula the effect of "accelerable" cycles in the CFG [10], and relies on widening for more complex cycles. *Guided static analysis* technique alternates ascending and descending sequences on an increasingly larger part of the system of equations [11]. This improves the accuracy of the analysis in many cases, but still it relies ultimately on the effectiveness of narrowing (see §2).

Widening with thresholds. Among local techniques, *widening up-to* or *widening with thresholds* attempts to bound the extrapolation performed by the standard widening ∇ operator [5,12]. The idea is to parameterize ∇ with a finite set $\mathcal{C}$ of *threshold constraints*, and to keep in the result $R = P \nabla_{\mathcal{C}} Q$ those constraints $c \in \mathcal{C}$ that are still satisfied by Q: $P \nabla_{\mathcal{C}} Q = (P \nabla Q) \sqcap \{c \in \mathcal{C} \mid Q \models c\}$. . Similarly to abstract acceleration techniques, widening with thresholds prevents

from going too high in the lattice of properties (see Fig. 1) and from propagating inaccurate invariants in the CFG of the program, which cannot be strengthened later by narrowing. However, the benefit provided by widening with thresholds fully depends on the choice of the thresholds.

Our contribution: thresholds inference. This paper develops a semantic-based technique to infer automatically relevant thresholds, by propagating constraints in the CFG of the program in an adequate way. §2 illustrates on small examples the strengths and weaknesses of widening and narrowing, and gives the rationale for our technique for inferring relevant thresholds, which is formalized in §3. §4 evaluates it on a number of example programs and compares it to guided static analysis [11] and policy iteration [3]. A longer version of this paper is available as a research report [13].

2 The Widening/Narrowing Approach in Practice

We assume a static analysis problem formalized as an equation system

$$X^{(k)} = F^{(k)}(X) \qquad X = (X^{(1)}, \ldots, X^{(K)}) \in C^K \tag{4}$$

where $X^{(k)} \in C$ is the concrete property associated with a node of the CFG of the program and $(C, \subseteq)$ is ordered by logical implication. Given an abstract domain $(A, \sqsubseteq)$ connected to C with a concretization function $\gamma : A \to C$, and a a *widening operator* $\nabla : A \times A \to A$ [6] we derive from (4) the system of equations

$$Y^{(k)} = G^{(k)}(Y) \qquad Y = (Y^{(1)}, \ldots, Y^{(K)}) \in A^K \tag{5}$$

In order to solve (5), we use chaotic iterations with widening [14]: we follow the iteration order $1 \ldots K$ and we apply widening as follows:

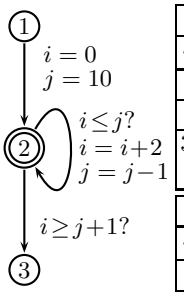
$$Y_0^{(k)} = \bot \qquad Y_{n+1}^{(k)} = \begin{cases} Y_n^{(k)} \nabla Y' & \text{if } k \in W \\ Y' & \text{otherwise} \end{cases} \tag{6}$$
$$\text{where } Y' = G^{(k)}(Y_{n+1}^{(0)} \ldots Y_{n+1}^{(k-1)}, Y_n^{(k)} \ldots Y_n^{(K)})$$

W is the subset of widening nodes: any dependency cycle in (5) contains a node in W. Narrowing by descending iteration is performed as in ((3)).

In all the examples of this paper, the static analysis problem is the computation of reachable values of the numerical variables of a program. A is the convex polyhedra domain, equipped with its standard widening operator [5].

Analysis of a simple loop program. Fig. 2 shows our first example. The double-line around a CFG node indicates a widening node in W. The table on the right details the Kleene iteration with widening and descending sequence, starting from $\bot$ at nodes ② and ③. In the steps 1 and 2, the widening operator has no effect. The row indexed by 3' corresponds to the computation of Y' in (6). In step 3, we have $Y_3^{(2)} = Y_2^{(2)} \nabla Y_{3'}^{(2)}$ and the effect of widening is to lose the upper bound on i. One descending step discovers the constraint $i \leq 26/3$, which comes from the postcondition of $Y_3^{(2)}$ by the loop:

```
var i,j:int;
begin
  i=0; j=10;
  while i<=j do
    i = i+2;
    j = j-1;
  done;
end
```



Ascending sequence		
n	$Y_n^{(2)}$	$Y_n^{(3)}$
1	$i=0 \wedge j=10$	
2	$i+2j=20 \wedge 0 \le i \le 2$	
$3'$	$i+2j=20 \wedge 0 \le i \le 4$	
3	$i+2j=20 \wedge 0 \le i$	$i+2j=20 \wedge 22 \le 3i$

Descending sequence		
n	$Z_n^{(2)}$	$Z_n^{(3)}$
1	$i+2j=20 \wedge 0 \le 3i \le 26$	$i+2j=20 \wedge 22 \le 3i \le 26$

Fig. 2. Example: single loop

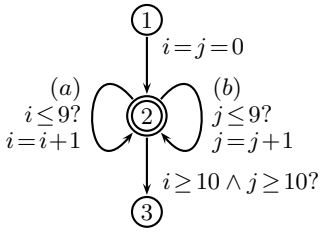


Fig. 3. Example: two non-deterministic loops

```
var i:int;
begin
  i=0;
  while true do
    if ? then
      i=i+1;
      if i>=100 then
        i=0;
  done;
end
```

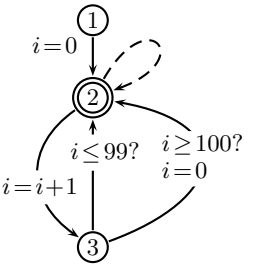


Fig. 4. Example: a single loop with break

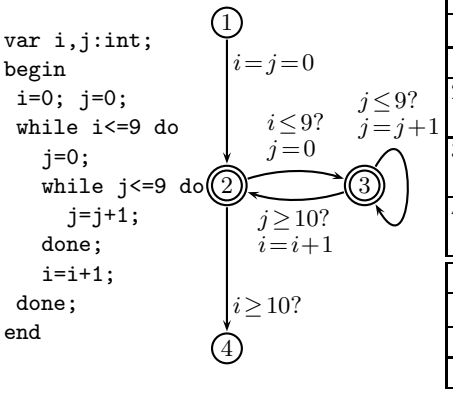
$$\exists i,j : \Big(\overbrace{i{+}2j{=}20}^{\text{implied by } Y_3^{(2)}} \wedge \overbrace{i \le j \wedge i' = i{+}2 \wedge j' = j{-}1}^{\text{loop transition}}\Big) = (i' = 20{-}2j' \wedge \boxed{i' \le j'{+}3}) \Rightarrow \underbrace{i' \le 20{-}2(i'{-}3)}_{= 3i' \le 26} \tag{7}$$

We first observe that the invariant $Z^{(3)}$ at point ③ can be rewritten into $i{+}2j{=}20 \wedge 8{-}\frac{2}{3} \le i \le 8{+}\frac{2}{3}$, so $i \le 26/3$ is the right bound for i at node ② Second, if one wants to use widening with thresholds, the guard of the loop $i \le j$ is not a useful threshold constraint. The effect of using this threshold constraint allows us to keep the constraint $i \le j$ at step 3, but this bound is violated at step $4'$ by the postcondition of the loop transition, hence this does not change the final result. We conclude that

(1) The important threshold constraint in a simple while loop is the postcondition of the guard of the loop by the loop body, here $i \le j{+}3$, see Eqn. (7).

Two non-deterministic loops. The CFG of Fig. 3 is typically the result of the asynchronous parallel product of two threads with a simple loop. It shows the limitation of descending sequences. The ascending sequence converges to $Y^{(2)} = 0 \le i \wedge 0 \le j$. The descending sequence fails to improve it:

$$\begin{aligned} Z_1^{(2)} &= G^{1\rightsquigarrow 2}(Y^{(1)}) \sqcup \quad G^{2\rightsquigarrow 2(a)}(Y^{(2)}) \quad \sqcup \quad G^{2\rightsquigarrow 2(b)}(Y^{(2)}) \\ &= \{i=j=0\} \sqcup \{1 \le i \le 10 \wedge 0 \le j\} \sqcup \{0 \le i \wedge 0 \le j \le 10\} \\ &= \{0 \le i \wedge 0 \le j\} \end{aligned}$$



Ascending sequence		
n	$Y_n^{(2)}$	$Y_n^{(3)}$
1	$i=j=0$	$i=j=0$
$2'$	$i=j=0$	$i=0 \wedge 0\leq j\leq 1$
2	$i=j=0$	$i=0 \wedge 0\leq j$
$3'$	$0\leq i\leq 1 \wedge 10i\leq j$	$0\leq i\leq 9 \wedge 0\leq j$
3	$0\leq i \quad \wedge 10i\leq j$	$0\leq i \quad \wedge 0\leq j$
$4'$	$0\leq i \quad \wedge 0\leq j$	$0\leq i \quad \wedge 0\leq j$
4	$0\leq i$	$0\leq i \quad \wedge 0\leq j$

Descending sequence		
n	$Z_n^{(2)}$	$Z_n^{(3)}$
1	$0\leq i \quad \wedge 0\leq j$	$0\leq i \quad \wedge 0\leq j\leq 10$
2	$j\leq 10i \wedge 0\leq j\leq 10$	$0\leq i \quad \wedge 0\leq j\leq 10$

Fig. 5. Example: nested loop

The problem is that, for both variables i and j, there is always one incoming edge in node ② that propagates an invariant without an upper bound on it. As a result, no variable gets an upper bound in the result.

A single loop with break. Another example, inspired by a real controller, is depicted on Fig. 4. The dashed self-loop comes from the non-deterministic test "?" modeling an input from the environment. When the "then" branch is not taken, nothing happens in the loop body. It makes the transfer function on node ② extensive: $G^{(2)}(Y) \sqsupseteq Y^{(2)}$. Hence, the descending sequence will never improve the invariant $Y^{(2)} = i \geq 0$ found by the ascending sequence.

Nested loop. The nested loop program of Fig. 5 contains two widening nodes ② and ③ and raises some additional issues. The ascending sequence loses the two constraints $j \leq 10$ (step 2) and $i \leq 10$ (step 3) as expected (it even loses $0 \leq j$ at step 4). The descending sequence first recovers $j \leq 10$ at point ③, but then fails to recover $i \leq 10$ at point ②. The problem is similar to the problem with the non-deterministic loops of Fig. 3:

- at point ②, the incoming edge ③ $\rightsquigarrow$ ② is not guarded by $i \leq 9$, and
- at point ③ the self-loop ③ $\rightsquigarrow$ ③ is also not guarded by $i \leq 9$.

Hence, $i \leq 10$ is neither recovered at node ② nor ③. On this example, the guided static analysis of [11] also fails to discover this bound. We observe that

(3) Applying the heuristics sketched at the end of single loop example for generating the threshold constraint, *i.e.,* considering the postcondition of the guard $i \leq 9$ by the body of the outer loop on i, already implies a fixpoint computation because of the inner loop on j.

(4) Once an important fact is lost and the induced approximation is propagated, it is not always possible to recover it with narrowing.

A loop with conditional and guided analysis. The example of Fig. 6 is taken from [11]. The loop proceeds in two phases: in the first one, i and j are incremented together until $i = 51$; in the second one, i is incremented and j is

decremented, and the loop exits with $i=102$ and $j=-1$. The standard approach finds, at node ④, $Y^{(4)} = j \leq -1 \wedge j \leq i+1$ and $Z_1^{(4)} = 51 \leq i \wedge j = -1$; it does not discover $i \leq 102$.

The intuition behind guided static analysis [11] is that widening implicitly assumes that the behavior of the program is "regular", which is not the case when a new behavior is activated in the program (in Fig. 6, such a new behavior is the activation of the "else" branch in the loop body). Hence its principle is (i) to discover the currently active part of the CFG (by a simple propagation); (ii) to perform a complete analysis with widening and narrowing on this part, starting from the invariants discovered so far; (iii) and to go back to step (i) to check whether new parts of the CFG may now be activated. The process is iterated up to convergence, which is guaranteed because the CFG is finite.

In this example, guided static analysis detects that only the "then" branch is initially activated. The ascending sequence on the active part of the CFG discovers $0 \leq i = j$ at node ② *followed by a descending sequence* that adds the bound $i \leq 51$. Only at this point does it take into account the activation of the "else" branch. The technique restarts a new analysis from the invariants inferred so far, and eventually obtains $Z_1^{(4)} = 51 \leq i \leq 102 \wedge j = -1$.

```
var i,j:int;
begin (1)
  i=0; j=0; (2)
  while true do
    if i<=50 then j=j+1;
             else j=j-1;
    if j<0 then goto (4)
    i=i+1;
  done; (4)
end
```

Fig. 6. Example: loop with conditional

In this example, widening with thresholds would behave like guided static analysis, *provided that the threshold constraint* $i \leq 51$ *is inferred.* Therefore,

(4) Thresholds are useful not only to bound $lfp(G)$, but also to *temporarily* bound the ascending iteration up to the activation of a new behavior.

Rationale for inferring thresholds. We made the following observations in the previous sections:

(1) For a while loop, the relevant threshold constraints are found in the post-condition of the guard of the loop by its body.
(2) Computing this postcondition may imply a fixpoint computation when the loop body itself contains loops; but then it implies widening.
(3) Threshold constraints inferred at a widening node should be propagated to the other widening nodes of the CFG.
(4) Thresholds are useful not only to bound the extrapolation, but also to detect the activation of new behaviors and to emulate guided analysis.

Because of observation (2), our solution propagates constraints without trying to converge to a fixpoint. Instead of the idea of propagating backward to the loop head the negation of the tests attached to transitions exiting a loop [15], our technique propagates forward the conditions for staying or exiting the loop body, which has a similar effect. In addition, *it also emulates guided analysis* by propagating tests attached to conditionals inside the loops.

3 Inferring Thresholds by Propagating Disjunctions

We assume the hypothesis of §2: we have to solve (4), which is abstracted in the abstract domain A into (5).

Definition 1 (Widening with thresholds). *Given two abstract values $a_1, a_2 \in A$, and a finite set $\mathcal{T} \subseteq A$ of* threshold values*, we define*

$$a_1 \nabla_{\mathcal{T}} a_2 = (a_1 \nabla a_2) \sqcap \bigsqcap \{a \in \mathcal{T} \mid a_1 \sqsubseteq a \wedge a_2 \sqsubseteq a\}$$

Extracting thresholds from an abstract property. We assume that we have an *extraction function* $\pi : A \to \wp(Elt(A))$ that extracts, from any value $a \in A$, a set of "threshold" abstract values $\{a_1, \ldots, a_t\}$ that satisfies $\forall i : a \sqsubseteq a_i$. The definition of π depends on the domain A and possibly on the widening operator ∇. For *numerical domains*, π typically extracts the set of numerical constraints on which abstract values are built by conjunction. For the logico-numerical domain BddApron [16], π also returns all the numerical constraints involved in the abstract property. π is extended to the disjunctive domain $\wp(A)$ with $\pi(X) = \bigcup_{a \in X} \pi(a)$.

Propagating thresholds in the system of equations. We now assume that (4) is abstracted into $\wp(A)$ rather than A. This can be done by replacing $\sqcup$ by $\cup$ inside the functions $G^{(k)}$ in (5). We thus have an equation system $T^{(k)} = G_d^{(k)}(T)$ with $T = (T^{(1)}, \ldots, T^{(K)}) \in (\wp(A))^K$. We also assume that, in the disjunctive domain $\wp(A)$, disjuncts are not simplified using the order $\sqsubseteq$ in A. We infer thresholds by considering the first steps of the following sequence:

$$\begin{array}{l} T_0^{(k)} = \top_{\wp(A)} = \{\top_A\} \\ T_{n+1}^{(k)} = \pi \circ G_d^{(k)}(T_{n+1}^{(0)} \ldots T_{n+1}^{(k-1)}, T_n^{(k)} \ldots T_n^{(K)}) \end{array} \tag{8}$$

Given a number N of iterations, we define the set $\mathcal{T}^{(k)}$ of threshold values attached to the node $k \in T$ as $\mathcal{T}^{(k)} = T_N^{(k)}$. In practice, we take $N = 2$. This allows us to propagate conditions from loop heads to each node of their body (first iteration) but also to propagate conditions of possible inner loops back to the head of the outer loops (second iteration).

Applying widening with thresholds. Finally we solve (5) by computing the sequence (6) in which $\nabla_{\mathcal{T}^{(k)}}$ replaces the standard widening operator ∇.

Application to the running examples. Figs 7 shows the application of our method to the examples described in §2. In each subfigure, the upper table shows the thresholds computed at each step while the lower table gives the result of the ascending sequence using thresholds. In all cases, the ascending sequence discovers the expected invariant. We do not break equality constraints $e = 0$ in $e \geq 0 \wedge e \leq$ during the inference of thresholds, but we do it at the end of the inference (in Fig. 7(d) the threshold $j \leq 10$ at node ② is extracted from the

n	$T_n^{(2)} \setminus T_{n-1}^{(2)}$
1	$\{\underline{i \leq j+3}, i=0, j=10\}$
2	$\{i \leq 12, i=2, j \geq -1, j=9\}$

k	2	3
$Y^{(k)}$	$i+2j=20 \wedge 3i \leq 26 \wedge i \geq 0$	$i+2j=20 \wedge 22 \leq 3i \leq 26$

(a) Single loop example of Fig. 2

n	$T_n^{(2)} \setminus T_{n-1}^{(2)}$
1	$\{\underline{i \leq 10}, \underline{j \leq 10}, i=0, j=10\}$
2	$\{i=1, j=1\}$

k	2	3
$Y^{(k)}$	$0 \leq i \leq 10 \wedge 0 \leq j \leq 10$	$i=10 \wedge j=10$

(b) Two non-deterministic loops example of Fig. 3

n	$T_n^{(2)} \setminus T_{n-1}^{(2)}$
1	$\{\underline{i \leq 99}, i=0, \top\}$
2	$\{i=1\}$

k	2	3
$Y^{(k)}$	$0 \leq i \leq 99$	$1 \leq i \leq 100$

(c) Single loop with break example of Fig. 4

n	$T_n^{(2)} \setminus T_{n-1}^{(2)}$	$T_n^{(3)} \setminus T_{n-1}^{(3)}$
1	$\{j \geq 10, i=0, j=0\}$	$\{\underline{i \leq 9}, \underline{j \leq 10}, i=0, j=0\}$
2	$\{\underline{i \leq 10}, \underline{j=10}, i=1\}$	$\{i=1, j=1\}$

k	2	3	4
$Y^{(k)}$	$j \leq 10i \wedge i \leq 10 \wedge 0 \leq j \leq 10$	$0 \leq i \leq 9 \wedge 0 \leq j \leq 10$	$i=10 \wedge j=10$

(d) Nested loops example of Fig. 5

Fig. 7. Inferring thresholds and widening with thresholds on running examples

Table 1. Comparison between standard, guided, policy iteration, and thresholds techniques using the box domain, on the examples of [3]

Program	**guided** vs **standard**	**policy** vs **guided**	**thresholds** vs **policy**
test5	=	4/0	=
test6	0/4	6/ − 4	0/4
test7	=	9/0	−4/0
test8	=	4/0	=
test9	2/0	4/0	=
test1, test2, test3, test4: same results (simple examples)			

N1/N2 in column A vs B: number N1 of *additional* finite interval bounds and number $N2$ of *improved* finite interval bounds found by technique A compared to technique B, in all the program CFG; **"="** indicates identical results.

value $j = 10$). Although our method infers many useless threshold constraints, it does infer all the required ones (which are underlined). It can be noticed that the second iteration step adds useful threshold constraints only in the nested loop example: this confirms observation (3) in §2.

4 Experiments and Conclusion

We implemented our inference technique for the BddApron logico-numerical abstract domain used by the ConcurInterproc tool [16,17].[1] We first consider the box abstract domain, and three alternative methods: (1) the **standard**

[1] These experiments can be run with the online version of the analyzer, see [17].

Table 2. Comparison between standard, guided and our technique (inference+analysis), using the convex polyhedra domain

Program	CFG Size	Standard		Guided		Inf. of Thres.		Thresholds	
	#K/#F	Time	Prec.	Time	Prec.	Time	Av.nb.	Time	Prec.
Sequential, intraprocedural programs									
loop1	3/3	0.02	=	0.03	=	0.02	14	0.02	=
loop_nondet	3/4	0.03	B	0.03	B	0.02	12	0.04	A
loop_reset	4/6	0.01	B	0.01	B	0.01	6	0.02	A
loop2	4/5	0.06	B	0.09	B	0.02	12	0.08	A
gopanreps	4/6	0.06	B	0.09	A	0.04	16	0.08	A
loop2Bis	5/7	0.14	B	0.24	B	0.07	18	0.20	A
gopanrepsBis	5/8	0.29	B	0.49	B	0.28	39	0.85	A
nestedLoop	5/8	0.61	B	0.68	B	0.58	39	0.72	A
sipma91	7/11	0.35	B	0.42	B	0.57	33	0.37	A
car	3/4	0.06	=	0.07	=	0.01	14	0.06	=
Concurrent programs									
concurrent_loop	9/16	0.04	B	0.04	B	0.07	8	0.05	A
loop2_TLM	24/26	0.24	B	0.25	B	1.63	19	0.33	A^+
barrier_counter_2	61/108	1.71	B	1.91	B	2.09	18	4.90	A^+
barrier_counter_3	405/847	158.00	B	190.00	B	1553.00	78	1096.00	A^+
Programs with non-inlined procedure calls									
loop2_rec	15/18	0.25	B	0.42	B	1.88	28	0.47	A
gopanreps_rec	9/11	0.22	B	0.38	A	2.17	46	0.46	A
loop2Bis_rec	16/20	1.07	B	1.74	B	23.75	43	1.25	A
gopanrepsBis_rec	17/21	3.29	B'	9.23	A	651.00	82	9.86	B"
loop2_TLM_rec	34/38	0.86	B	0.86	B	17.76	20	1.97	A^+

#K/#F: size of the CFG, with #K the number of control nodes and #F the number of basic blocks; **Time:** running times in seconds, on a MacBook Air (Intel Core 2 Duo, 2.13 GHz); **Prec.:** relative precision: A is best, C is worse; A^+ indicates the proof of a specific property; **Av.nb.:** average number of inferred threshold constraints at each CFG node.

Kleene iteration with widening and descending sequence; (2) the **guided** static analysis technique of [11]; (3) and the **policy** iteration technique of [3] mentioned in the introduction, which is able to converge to the least fixpoint under some assumptions. Tab. 2 compares the results of the 4 methods on the examples of [3], which are purely numerical, by counting the total number of better bounds inferred by one technique over the other. On these tricky examples:

- **guided** is always better than **standard**;
- **policy** is better than **guided**, with the exception of **test6**, where it infers 6 additional finite bounds, but where 4 of the other inferred bounds are less accurate. **Thresholds** does strictly better than the other techniques here.
- **test7** is the only example for which widening with thresholds is less accurate than policy iteration, but still more accurate that guided analysis.

These experiments showed us the usefulness of considering also the constraint $x \geq 0$ when $x \leq 0$ is inferred. Typically, if we have a *inner* loop `while (x>=1) do x-`, the exit constraint $x \leq 0$ will be propagated to the *outer* loop head, whereas it is the constraint $x \geq 0$ which is relevant as a threshold at this point.

We then considered the convex polyhedra abstract domain combined in BddApron with finite-state variables, Tab. 1. Policy iteration could not be experimented, because it is not defined on convex polyhedra. For all but 5 of these examples, widening with thresholds is strictly more precise than the standard or guided analyses, and it is less precise than guided analysis for a single example. W.r.t. efficiency, for the **sequential, simple examples**, the additional cost can be considered moderate, even when the number of inferred thresholds is not so small; for **concurrent programs**, the additional complexity is higher and may be dramatic in some cases, typically **barrier_counter_3** for which the number of thresholds have an impact of the analysis time (factor 6.0 w.r.t. standard analysis, besides the inference time). The performance problem here can be fixed by performing a thread-modular inference, which would infer the required thresholds on these examples (checked by manual inspection); for **relational interprocedural analysis**, we also have a performance problem, which results from the procedure return operation that implies a relation composition between abstract values. This problem desserves further investigations. Observe however that the technique infers the right thresholds, when for instance nested loops are implemented as tail recursive calls (**X_rec** versions of **X** examples).

To conclude, our technique is very successful w.r.t. precision, but needs efficiency improvements for concurrent and recursive programs. Abstract acceleration [10] might be better than our technique because it computes $\alpha \circ F^* \circ \gamma$ instead of the less precise $(\alpha \circ F \circ \gamma)^*$, but it does not solve the **nestedLoop** example with 3 nested loops, and is hardly applicable if loops are transformed in tail-recursive calls. It should combine efficiently with our technique. [18] describes the inference of thresholds in the Astrée analyzer; it infers thresholds for single variables, and considers intraprocedural programs (procedures are inlined).

References

1. Su, Z., Wagner, D.: A class of polynomially solvable range constraints for interval analysis without widenings and narrowings. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 280–295. Springer, Heidelberg (2004)
2. Gawlitza, T., Leroux, J., Reineke, J., Seidl, H., Sutre, G., Wilhelm, R.: Polynomial precise interval analysis revisited. In: Albers, S., Alt, H., Näher, S. (eds.) EA 2009. LNCS, vol. 5760, pp. 422–437. Springer, Heidelberg (2009)
3. Costan, A., Gaubert, S., Goubault, É., Martel, M., Putot, S.: A policy iteration algorithm for computing fixed points in static analysis of programs. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 462–475. Springer, Heidelberg (2005)
4. Gawlitza, T., Seidl, H.: Precise relational invariants through strategy iteration. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 23–40. Springer, Heidelberg (2007)

5. Halbwachs, N., Proy, Y., Roumanoff, P.: Verification of real-time systems using linear relation analysis. Formal Methods in System Design 11 (1997)
6. Cousot, P., Cousot, R.: Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In: Bruynooghe, M., Wirsing, M. (eds.) PLILP 1992. LNCS, vol. 631, pp. 269–295. Springer, Heidelberg (1992)
7. Miné, A.: The octagon abstract domain. In: Higher-Order and Symbolic Computation, vol. 19 (2006)
8. Bagnara, R., Hill, P.M., Ricci, E., Zafanella, E.: Precise widening operators for convex polyhedra. In: Science of Computer Programming (SCP), vol. 58 (2005)
9. Simon, A., Chen, L.: Simple and precise widenings for polyhedra. In: Ueda, K. (ed.) APLAS 2010. LNCS, vol. 6461, pp. 139–155. Springer, Heidelberg (2010)
10. Gonnord, L., Halbwachs, N.: Combining widening and acceleration in linear relation analysis. In: Yi, K. (ed.) SAS 2006. LNCS, vol. 4134, pp. 144–160. Springer, Heidelberg (2006)
11. Gopan, D., Reps, T.W.: Guided static analysis. In: Riis Nielson, H., Filé, G. (eds.) SAS 2007. LNCS, vol. 4634, pp. 349–365. Springer, Heidelberg (2007)
12. Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Rival, X.: Why does astrée scale up? Formal Methods in System Design 35 (2009)
13. Lakhdar-Chaouch, L., Jeannet, B., Girault, A.: Widening with thresholds for programs with complex control graphs. Rapport de recherche RR-7673, INRIA (2011)
14. Bourdoncle, F.: Efficient chaotic iteration strategies with widenings. In: Pottosin, I.V., Bjorner, D., Broy, M. (eds.) FMP&TA 1993. LNCS, vol. 735, pp. 128–141. Springer, Heidelberg (1993)
15. Halbwachs, N.: A heuristics for inferring thresholds (personal communication)
16. Jeannet, B.: The BDDAPRON logico-numerical abstract domains library, `http://www.inrialpes.fr/pop-art/people/bjeannet/bjeannet-forge/bddapron/`
17. Jeannet, B.: The ConcurInterproc interprocedural analyzer for concurrent programs, `http://pop-art.inrialpes.fr/interproc/concurinterprocweb.cgi`
18. Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: Combination of abstractions in the astrée static analyzer. In: Okada, M., Satoh, I. (eds.) ASIAN 2006. LNCS, vol. 4435, pp. 272–300. Springer, Heidelberg (2008)